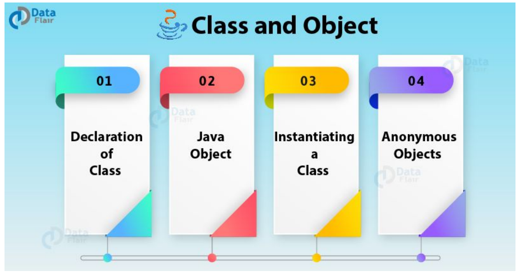
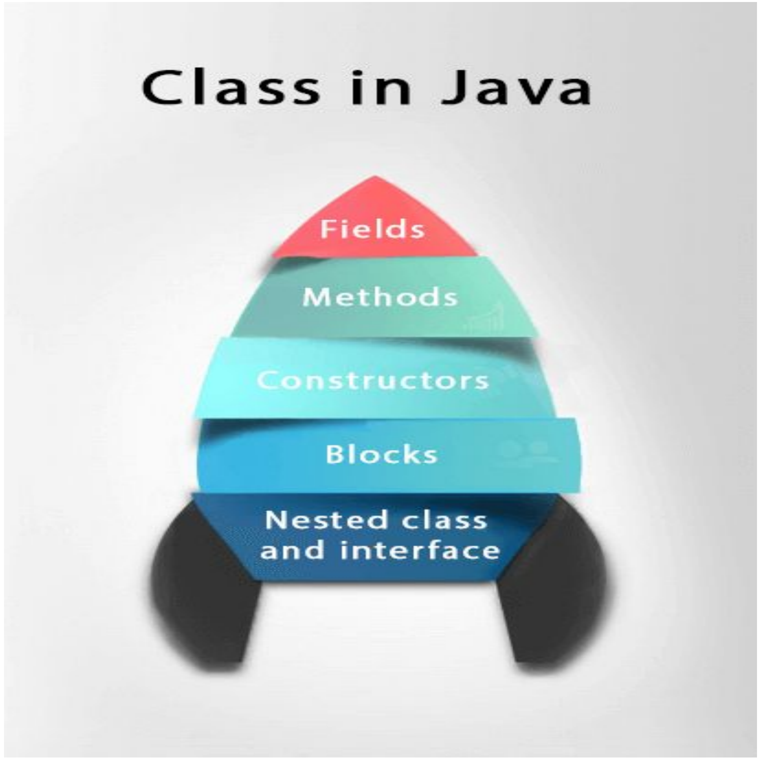
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

CHAPTER 4
OBJECTS AND CLASSES



SUBJECT:OOP-I CODE:3140705	PREPARED BY: ASST.PROF.NENSI KANSAGARA (CSE DEPARTMENT,ACET)	AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

DEFINING CLASSES FOR OBJECTS



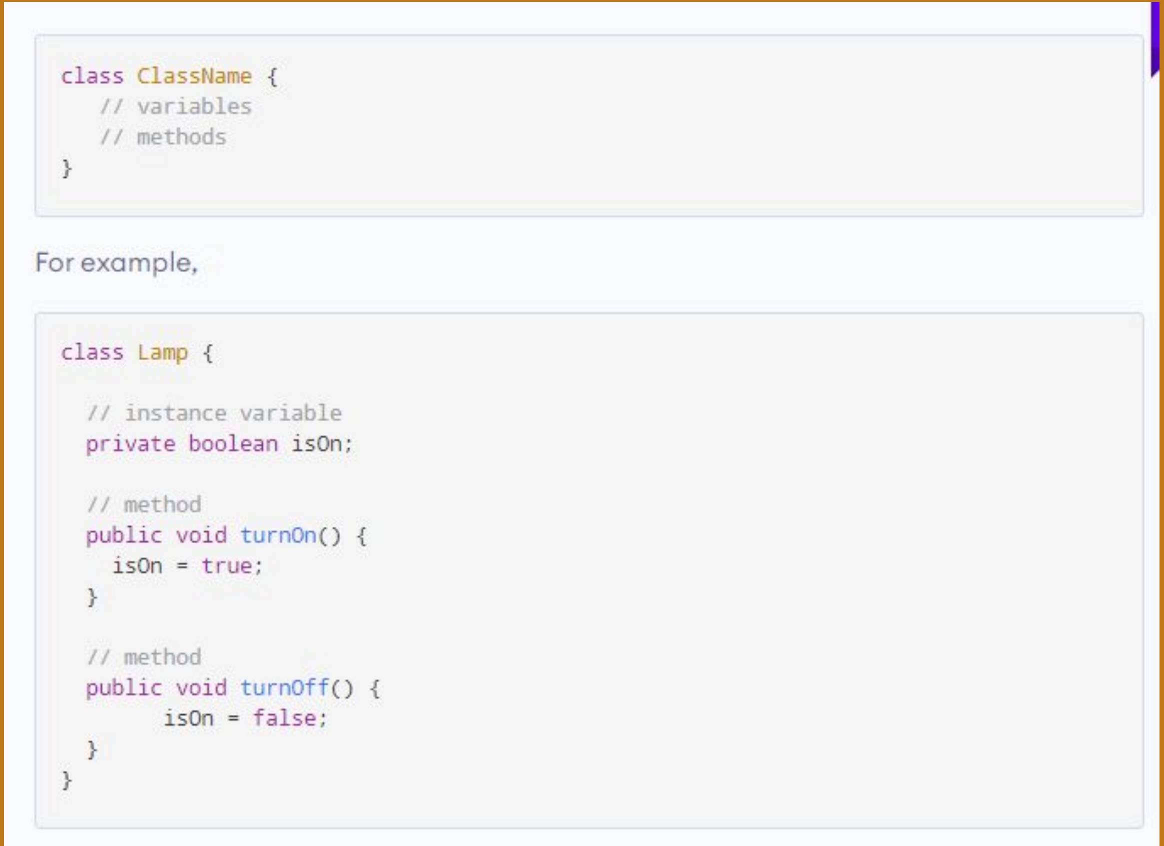
CLASS:

- ❖ A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

- **Modifiers** : A class can be public or has default access (Refer this for details).
- **Class name:** The name should begin with a initial letter (capitalized by convention).
- **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
- **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
- **Body:** The class body surrounded by braces, { }.

HOW TO DEFINE CLASS?



```
class ClassName {
   // variables
   // methods
}
```

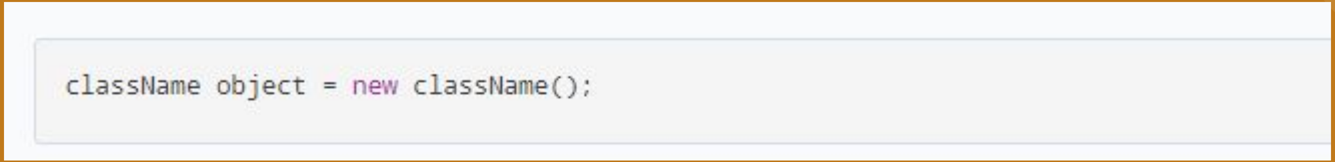
For example,

```
class Lamp {

  // instance variable
  private boolean isOn;

  // method
  public void turnOn() {
    isOn = true;
  }

  // method
  public void turnOff() {
        isOn = false;
  }
}
```

OBJECTS

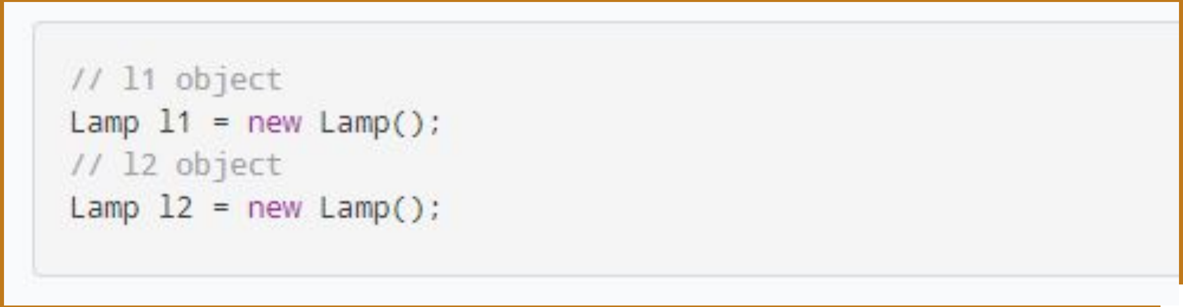
- ❖ It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :
  - ➢ **State** : It is represented by attributes of an object. It also reflects the properties of an object.
  - ➢ **Behavior** : It is represented by methods of an object. It also reflects the response of an object with other objects.
  - ➢ **Identity** : It gives a unique name to an object and enables one object to interact with other objects.
- ❖ Example of an object : dog

HOW TO DEFINE OBJECT?



```
className object = new className();
```

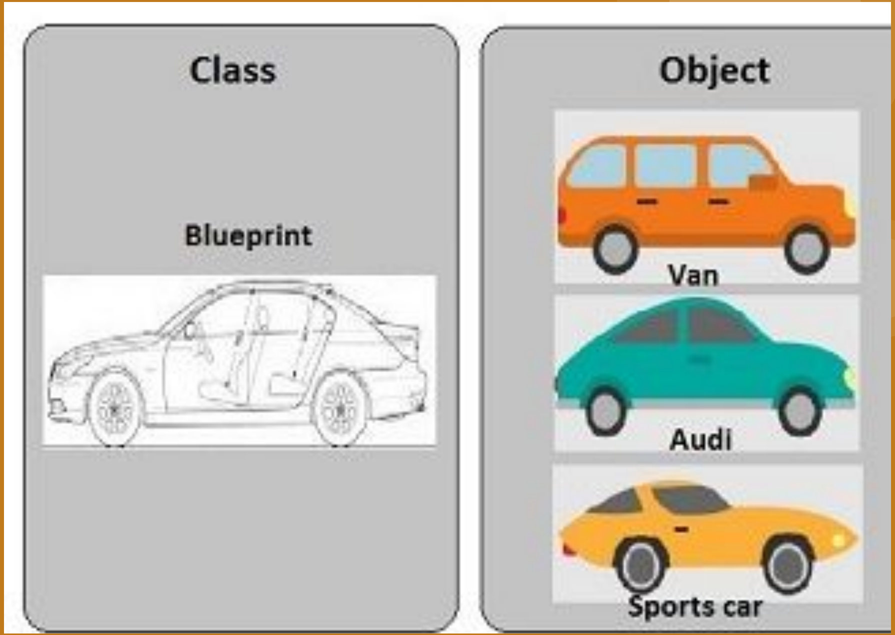
EXAMPLE:

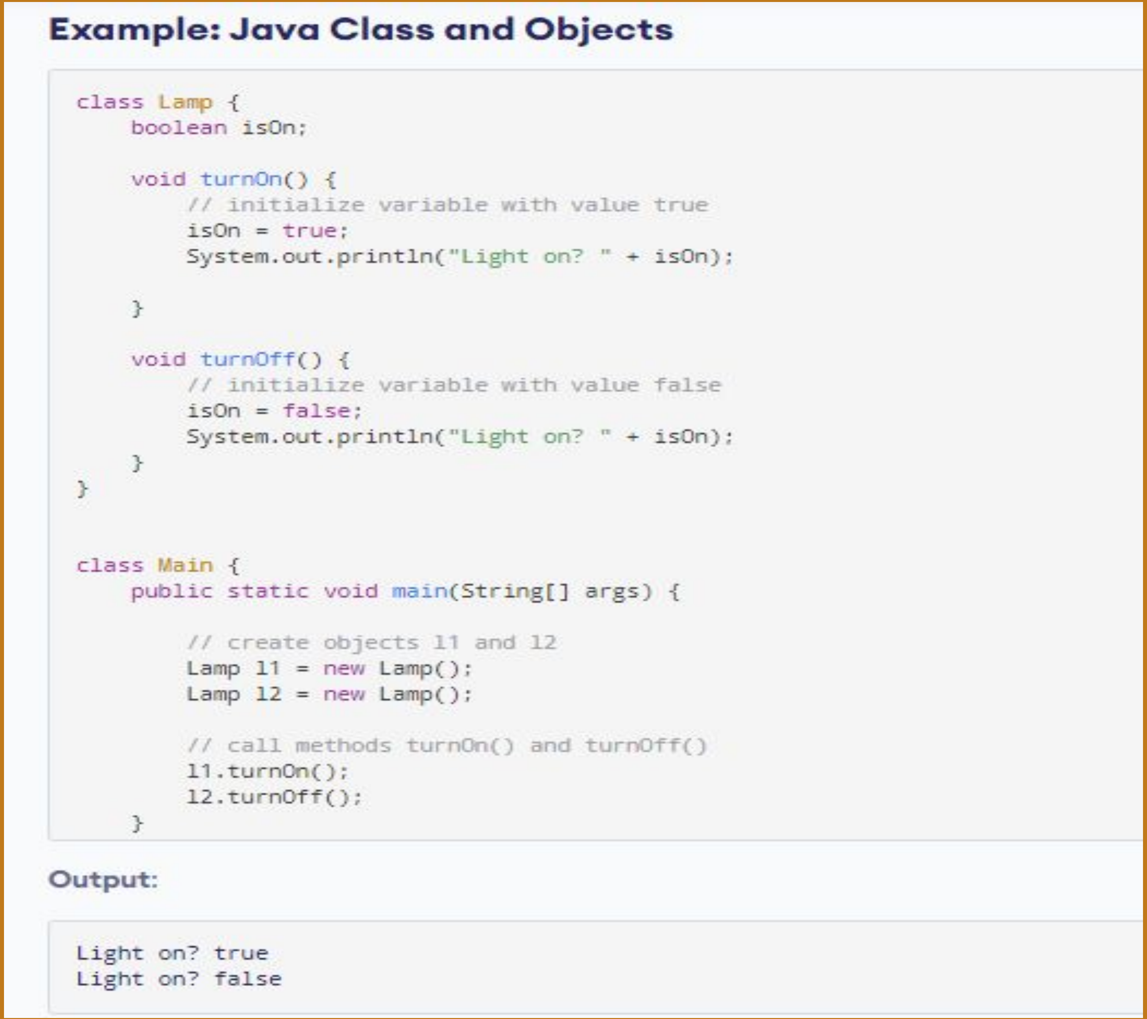


```
// l1 object
Lamp l1 = new Lamp();
// l2 object
Lamp l2 = new Lamp();
```

Data Name
Data Field
Methods

Class Template





Example: Java Class and Objects

```java
class Lamp {
    boolean isOn;

    void turnOn() {
        // initialize variable with value true
        isOn = true;
        System.out.println("Light on? " + isOn);

    }

    void turnOff() {
        // initialize variable with value false
        isOn = false;
        System.out.println("Light on? " + isOn);
    }
}


class Main {
    public static void main(String[] args) {

        // create objects l1 and l2
        Lamp l1 = new Lamp();
        Lamp l2 = new Lamp();

        // call methods turnOn() and turnOff()
        l1.turnOn();
        l2.turnOff();
    }
```

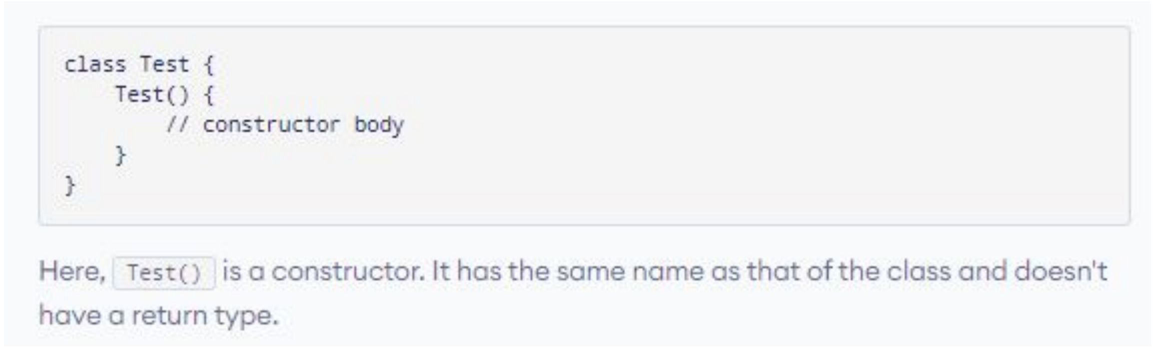
Output:

```
Light on? true
Light on? false
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

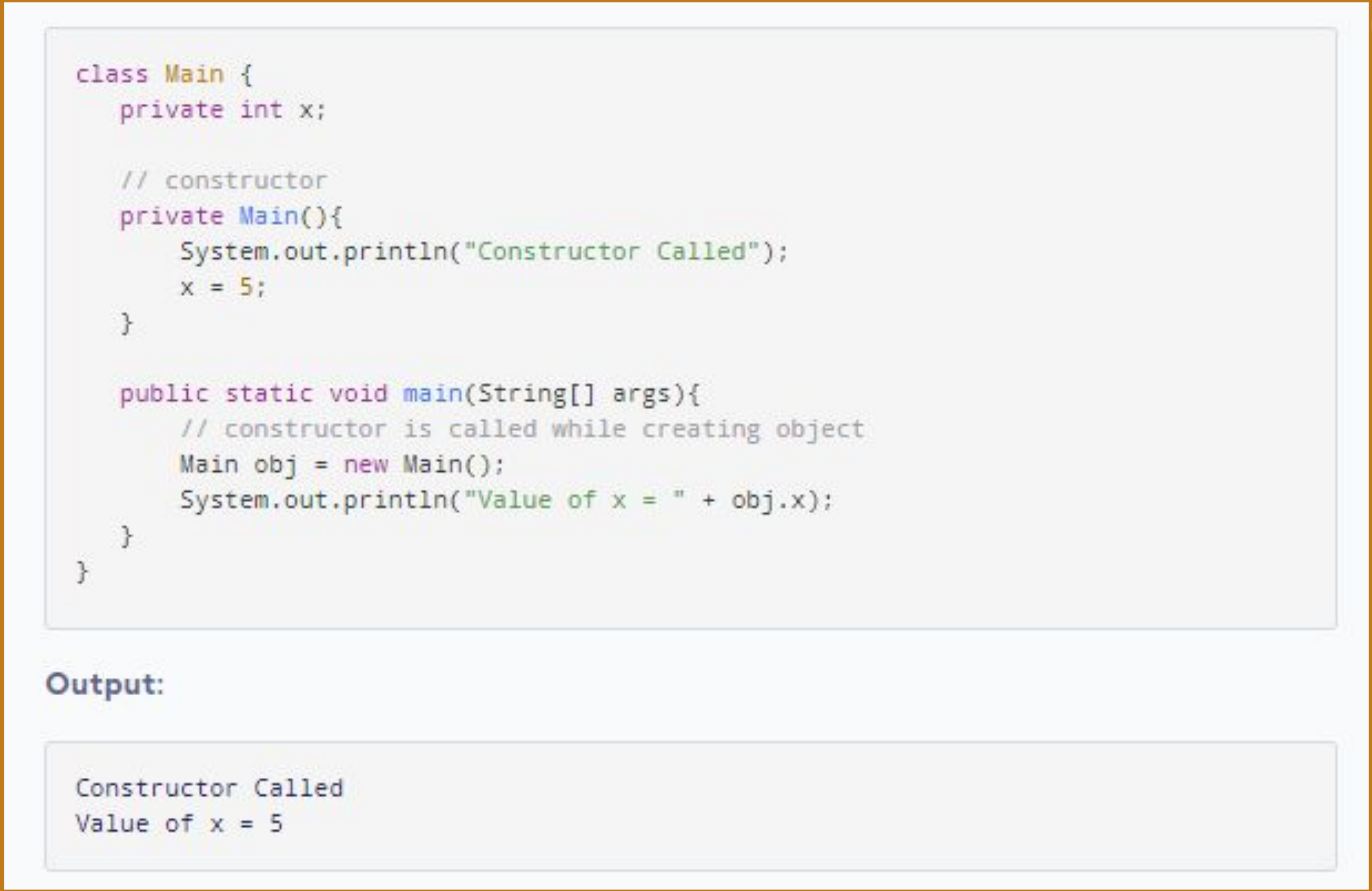
CONSTRUCTORS

- ❖ In Java, every class has its constructor that is invoked automatically when an object of the class is created. A constructor is similar to a method but in actual, it is not a method.
- ❖ A Java method and Java constructor can be differentiated by its name and return type. A constructor has the same name as that of class and it does not return any value. For



```
class Test {
    Test() {
        // constructor body
    }
}
```

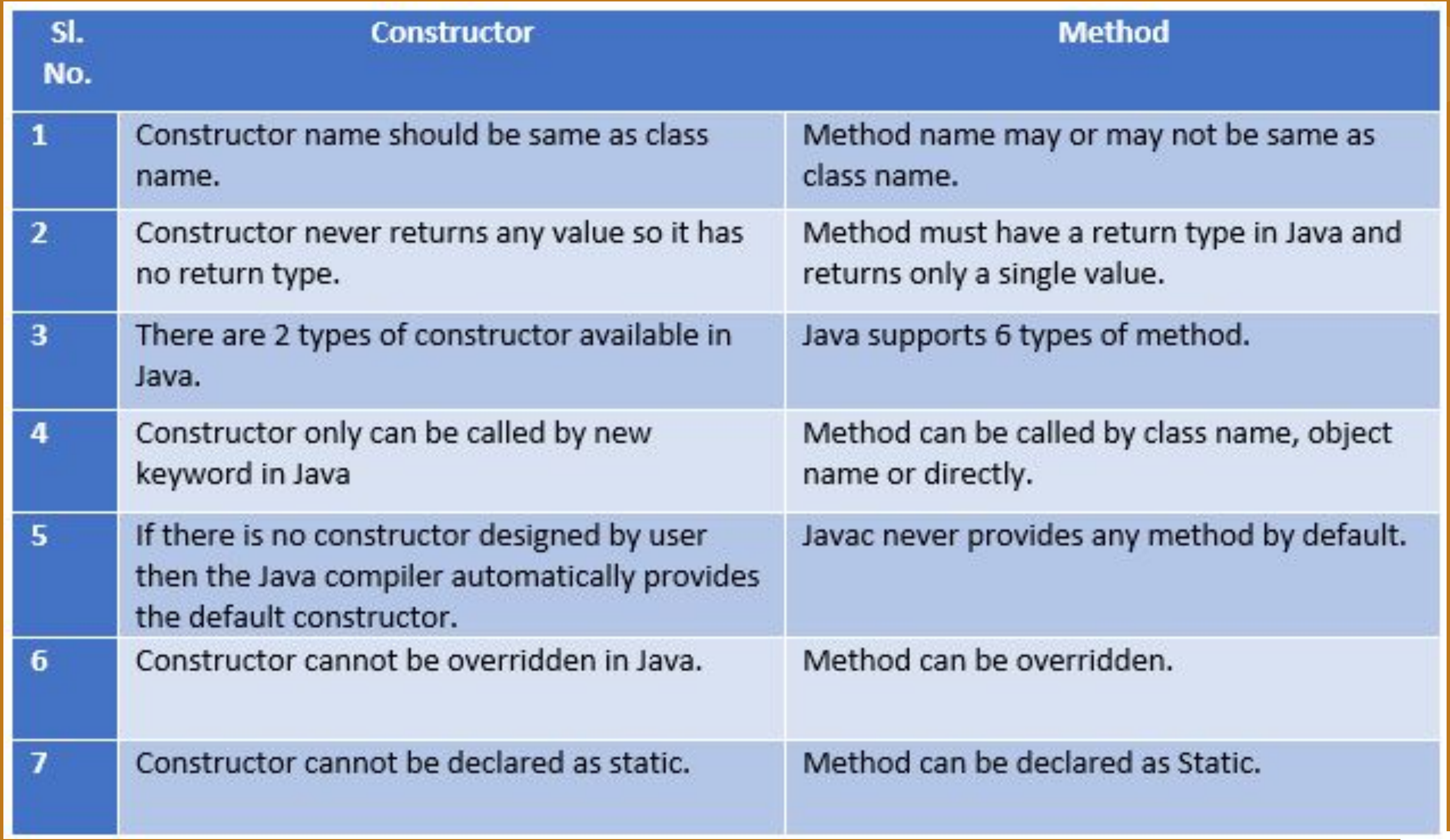
Here, `Test()` is a constructor. It has the same name as that of the class and doesn't have a return type.



```
class Main {
   private int x;

   // constructor
   private Main(){
       System.out.println("Constructor Called");
       x = 5;
   }

   public static void main(String[] args){
       // constructor is called while creating object
       Main obj = new Main();
       System.out.println("Value of x = " + obj.x);
   }
}
```

Output:

```
Constructor Called
Value of x = 5
```



Sl. No.	Constructor	Method
1	Constructor name should be same as class name.	Method name may or may not be same as class name.
2	Constructor never returns any value so it has no return type.	Method must have a return type in Java and returns only a single value.
3	There are 2 types of constructor available in Java.	Java supports 6 types of method.
4	Constructor only can be called by new keyword in Java	Method can be called by class name, object name or directly.
5	If there is no constructor designed by user then the Java compiler automatically provides the default constructor.	Javac never provides any method by default.
6	Constructor cannot be overridden in Java.	Method can be overridden.
7	Constructor cannot be declared as static.	Method can be declared as Static.

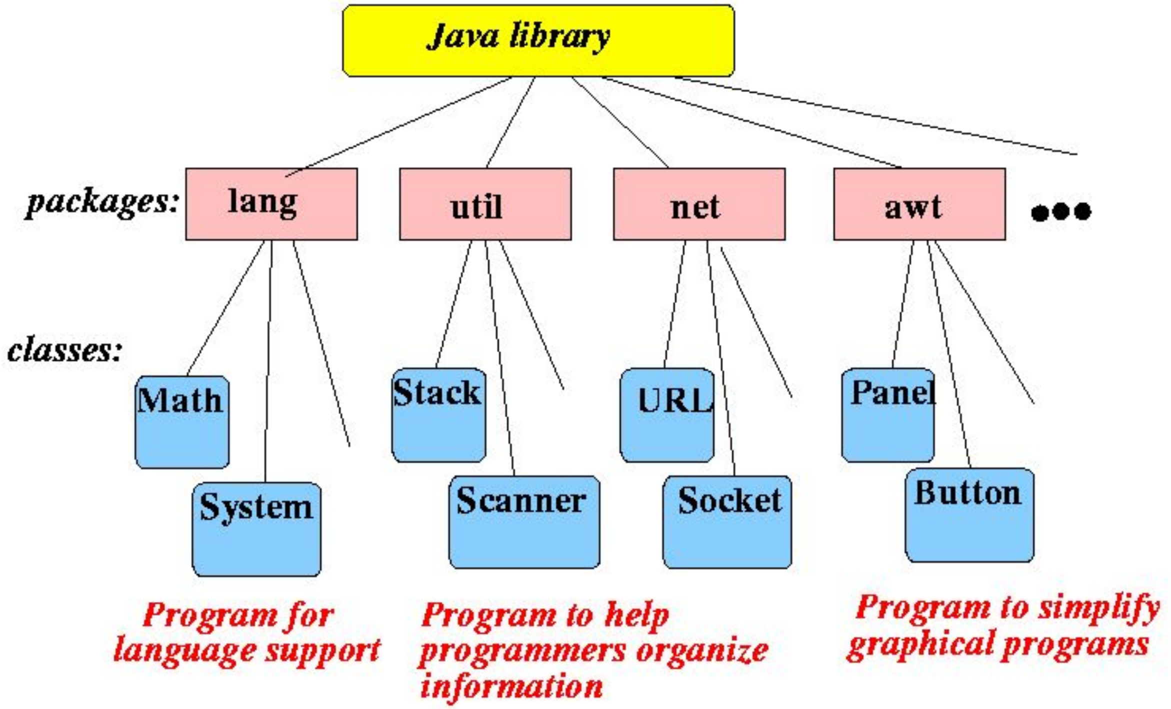
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

ACCESSING OBJECTS VIA REFERENCE VARIABLE

- ❖ The only way you can **access** an **object** is **through** a **reference variable**. A **reference variable** is declared to be of a specific type and that type can never be changed. **Reference variables** can be declared as static **variables**, instance **variables**, method parameters, or local **variables**.

- **Reference Variable Syntax:**
  - Class_name Object_Refrence_Varaible =new Class_name();
- **Example**
  - Student S= new Student();
- **Syntax of accessing Object’s member:**
  - Object_variable_name.object_member;

CLASSES FROM JAVA LIBRARY


Java library
packages:
lang
util
net
awt
●●●
classes:
Math
System
Stack
Scanner
URL
Socket
Panel
Button
Program for language support
Program to help programmers organize information
Program to simplify graphical programs

- ❖ The type constraints are used to control where the java library classes can be replaced with routine versions without affecting type perfection of programs.
- ❖ Static analysis is then used to control those applicants for which unused library functionality and synchronization can be removed safely from the allocated types.
- ❖ The profile data is collected about the usage features of the customization candidates to determine where the allocation of custom library classes is likely to be cost-effective.
- ❖ To base on the static analysis results and the profiling information the custom library classes are automatically generated from a template.
- ❖ The bytecode of the client application is rewritten to use the generated custom classes. This bytecode rewriting is completely see-through to the programmer.

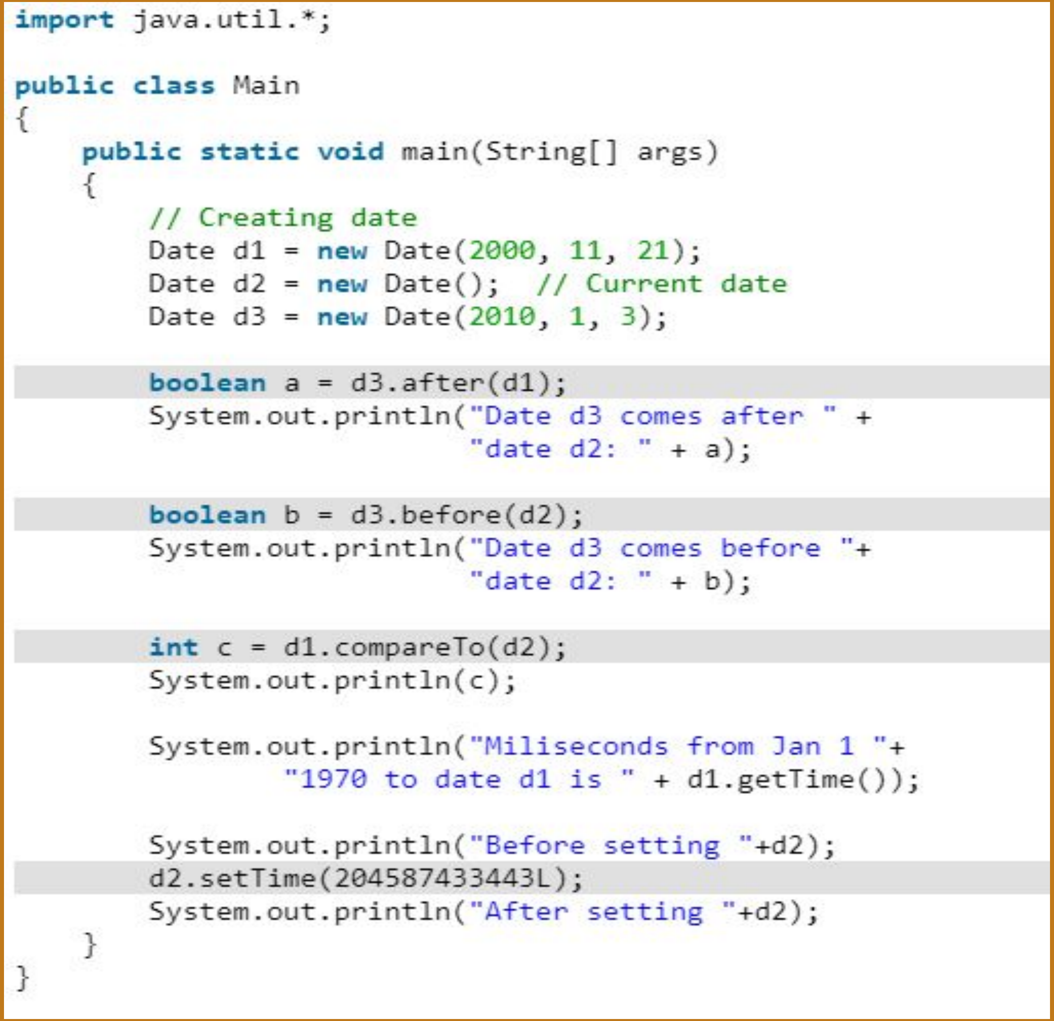
DATE CLASS

- The **Date class** of **java**. util package implements Serializable, Cloneable and Comparable interface. It provides constructors and methods to deal with **date** and time with **java**. **Date**() : Creates **date** object representing current **date** and time.
- **Constructors**
  - **Date()** : Creates date object representing current date and time.
  - **Date(long milliseconds)** : Creates a date object for the given milliseconds since January 1, 1970, 00:00:00 GMT.
  - **Date(int year, int month, int date)**
  - **Date(int year, int month, int date, int hrs, int min)**
  - **Date(int year, int month, int date, int hrs, int min, int sec)**
  - **Date(String s)**

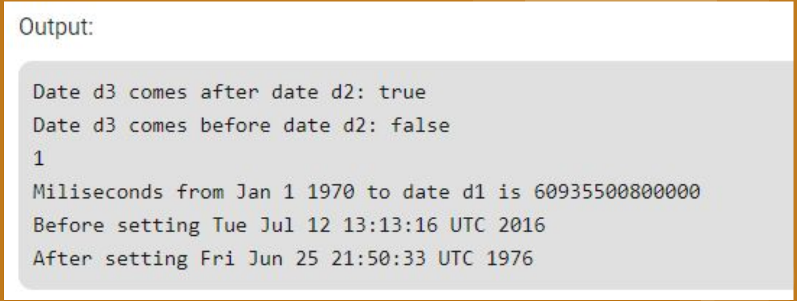
❖ **Important Methods**

- **boolean after(Date date) :** Tests if current date is after the given date.
- **boolean before(Date date) :** Tests if current date is before the given date.
- **int compareTo(Date date) :** Compares current date with given date. Returns 0 if the argument Date is equal to the Date; a value less than 0 if the Date is before the Date argument; and a value greater than 0 if the Date is after the Date argument.
- **long getTime()** : Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this Date object.
- **void setTime(long time)** : Changes the current date and time to given time.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY



```java
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        // Creating date
        Date d1 = new Date(2000, 11, 21);
        Date d2 = new Date();  // Current date
        Date d3 = new Date(2010, 1, 3);

        boolean a = d3.after(d1);
        System.out.println("Date d3 comes after " +
                           "date d2: " + a);

        boolean b = d3.before(d2);
        System.out.println("Date d3 comes before "+
                           "date d2: " + b);

        int c = d1.compareTo(d2);
        System.out.println(c);

        System.out.println("Miliseconds from Jan 1 "+
                "1970 to date d1 is " + d1.getTime());

        System.out.println("Before setting "+d2);
        d2.setTime(204587433443L);
        System.out.println("After setting "+d2);
    }
}
```



Output:

```
Date d3 comes after date d2: true
Date d3 comes before date d2: false
1
Miliseconds from Jan 1 1970 to date d1 is 60935500800000
Before setting Tue Jul 12 13:13:16 UTC 2016
After setting Fri Jun 25 21:50:33 UTC 1976
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

RANDOM CLASS

- ❖ Random class is used to generate pseudo-random numbers in java. An instance of this class is thread-safe. The instance of this class is however cryptographically insecure. This class provides various method calls to generate different random data types such as float, double, int.
- ❖ **Constructors:**
  - ➢ **Random()**: Creates a new random number generator
  - ➢ **Random(long seed)**: Creates a new random number generator using a single long seed

DECLARATION

```
public class Random
        extends Object
         implements Serializable
```

1)java.util.Random.doubles(): Returns an effectively unlimited stream of pseudo random double values, each between zero (inclusive) and one (exclusive)

Syntax:

public DoubleStream doubles()

Returns:

a stream of pseudorandom double values

2)java.util.Random.ints(): Returns an effectively unlimited stream of pseudo random int values

Syntax:

public IntStream ints()

Returns:

a stream of pseudorandom int values

3)java.util.Random.longs(): Returns an effectively unlimited stream of pseudo random long values
Syntax:
public LongStream longs()

Returns:

a stream of pseudorandom long values

4)next(int bits): java.util.Random.next(int bits) Generates the next pseudo random number
Syntax:
protected int next(int bits)

Parameters:

bits - random bits

Returns:

the next pseudo random value

5)java.util.Random.nextBoolean(): Returns the next pseudo random, uniformly distributed boolean value from this random number generator’s sequence

Syntax:

public boolean nextBoolean()

Returns:

the next pseudorandom, uniformly distributed boolean value

1. from this random number generator's sequence

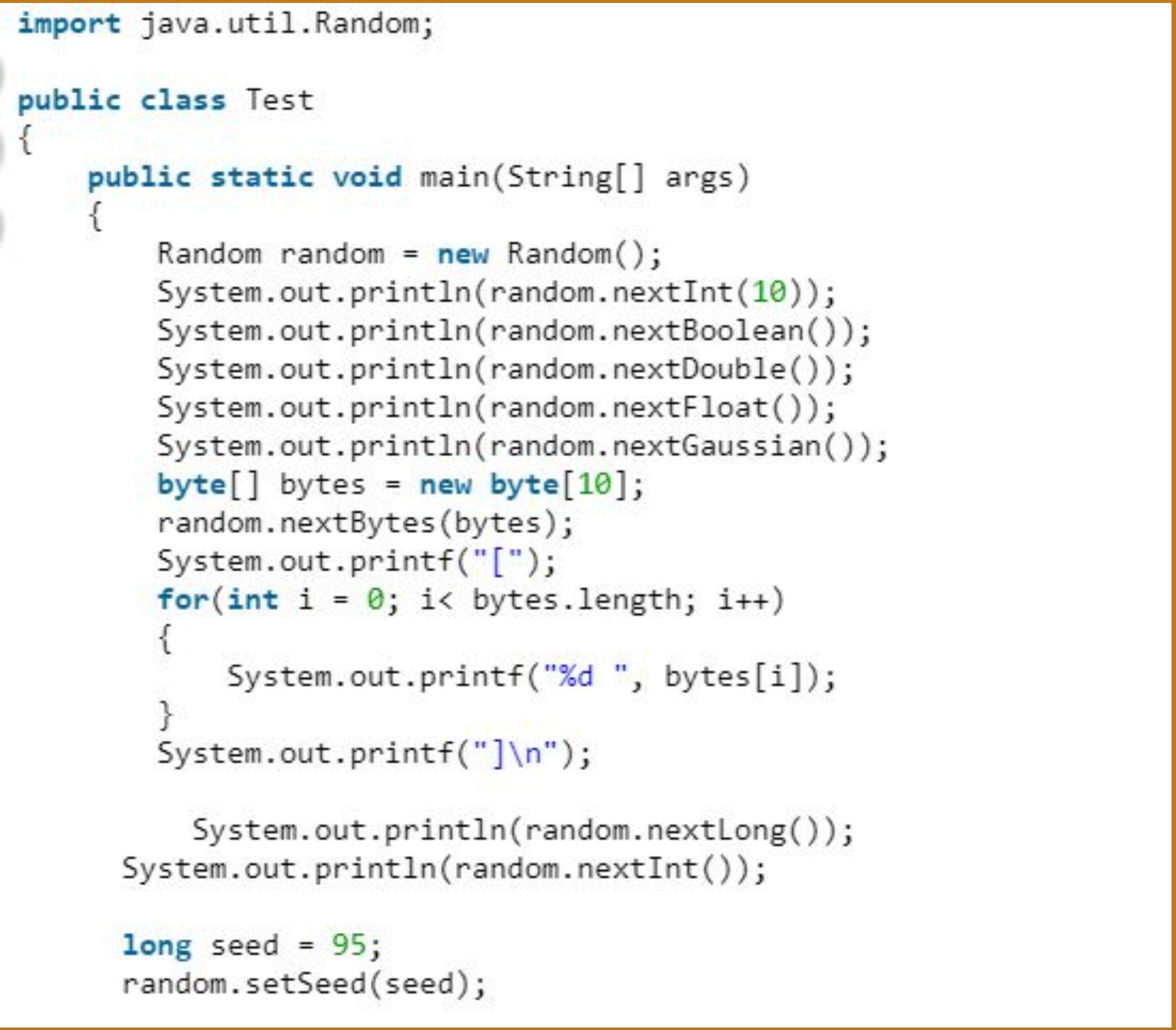
6)java.util.Random.nextBytes(byte[] bytes) : Generates random bytes and places them into a user-supplied byte array

Syntax:

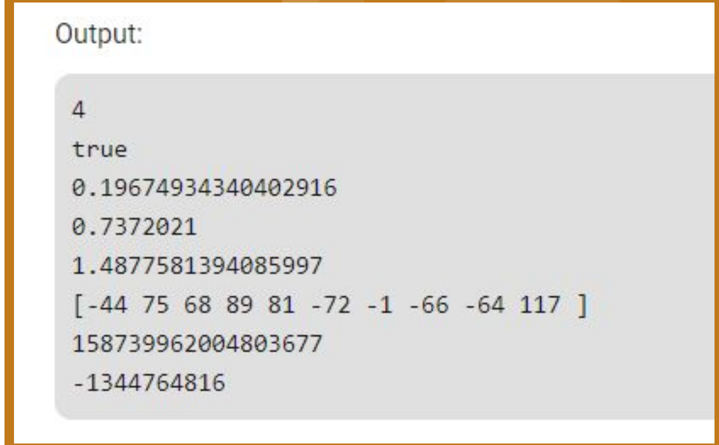
public void nextBytes(byte[] bytes)

Parameters:

bytes - the byte array to fill with random bytes



```java
import java.util.Random;

public class Test
{
    public static void main(String[] args)
    {
        Random random = new Random();
        System.out.println(random.nextInt(10));
        System.out.println(random.nextBoolean());
        System.out.println(random.nextDouble());
        System.out.println(random.nextFloat());
        System.out.println(random.nextGaussian());
        byte[] bytes = new byte[10];
        random.nextBytes(bytes);
        System.out.printf("[");
        for(int i = 0; i< bytes.length; i++)
        {
            System.out.printf("%d ", bytes[i]);
        }
        System.out.printf("]\n");

          System.out.println(random.nextLong());
      System.out.println(random.nextInt());

      long seed = 95;
      random.setSeed(seed);
```



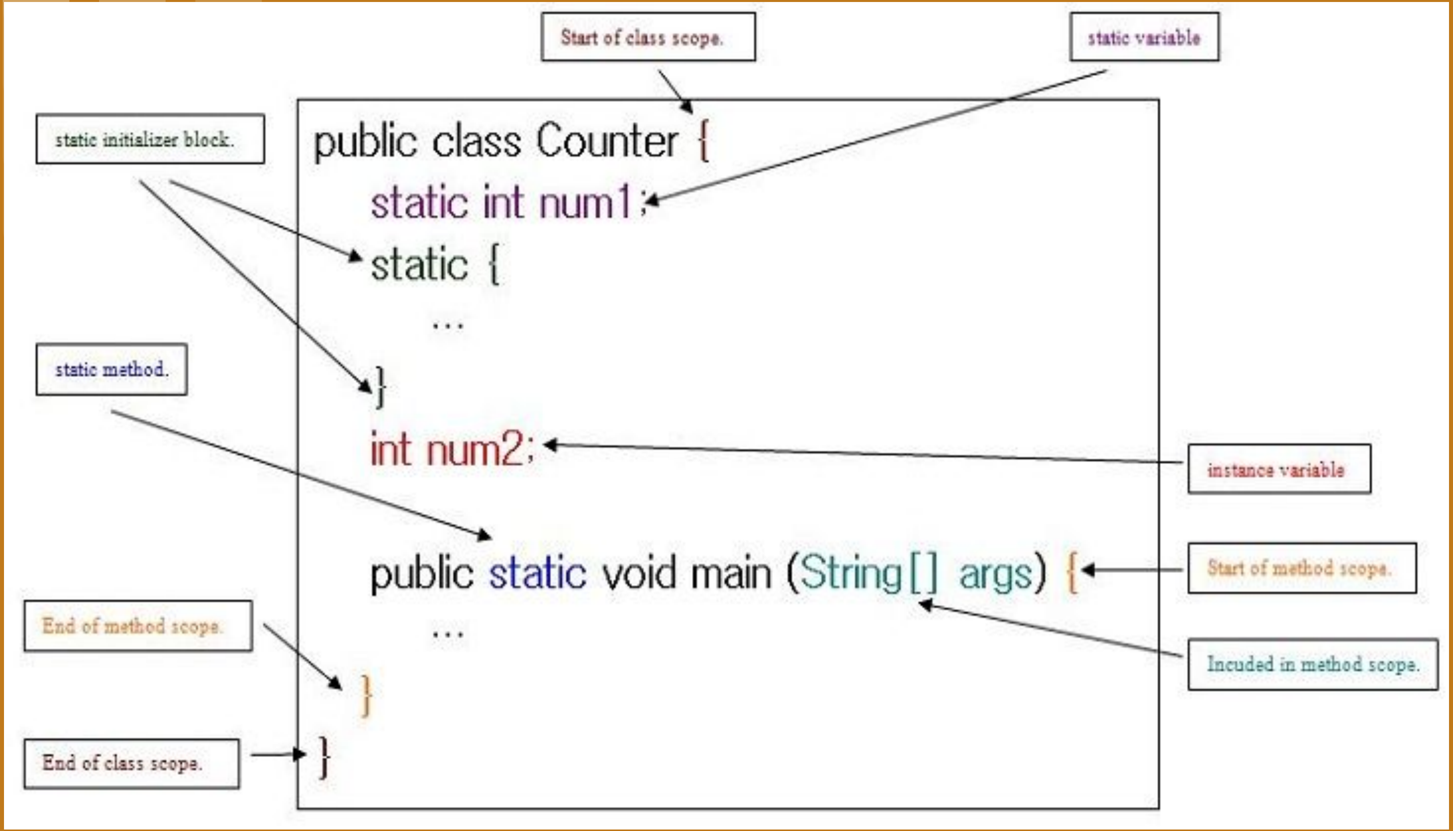
Output:

```
4
true
0.19674934340402916
0.7372021
1.4877581394085997
[-44 75 68 89 81 -72 -1 -66 -64 117 ]
158739962004803677
-1344764816
```

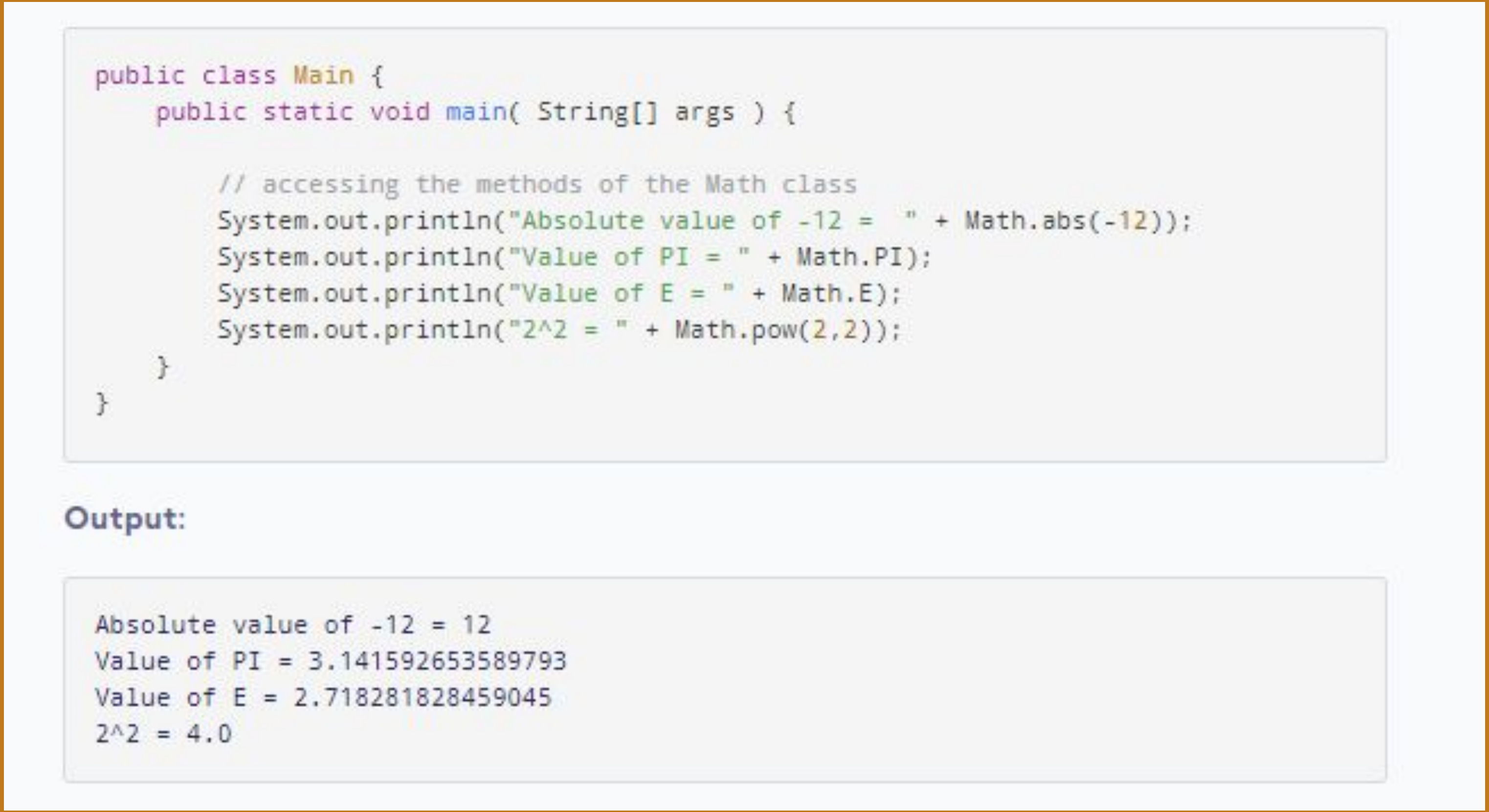
STATIC VARIABLE,METHODS AND CONSTANTS

- ❖ In Java, if we want to access class members, we must first create an instance of the class. But there will be situations where we want to access class members without creating any variables.
- ❖ In those situations, we can use the static keyword in Java. If we want to access class members without creating an instance of the class, we need to declare the class members static.
- ❖ The Math class in Java has almost all of its members static. So, we can access its members without creating instances of the Math class. For example,

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY



```
public class Counter {
    static int num1;
    static {
        ...
    }
    int num2;

    public static void main (String[] args) {
        ...
    }
}
```



```java
public class Main {
    public static void main( String[] args ) {

        // accessing the methods of the Math class
        System.out.println("Absolute value of -12 =  " + Math.abs(-12));
        System.out.println("Value of PI = " + Math.PI);
        System.out.println("Value of E = " + Math.E);
        System.out.println("2^2 = " + Math.pow(2,2));
    }
}
```

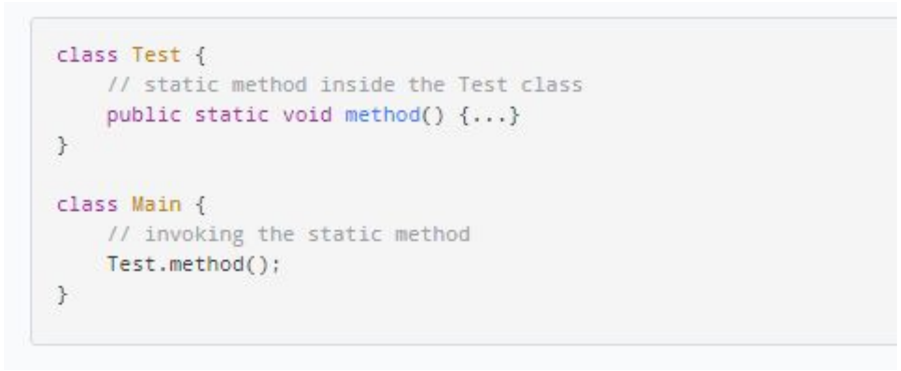
Output:

```
Absolute value of -12 = 12
Value of PI = 3.141592653589793
Value of E = 2.718281828459045
2^2 = 4.0
```

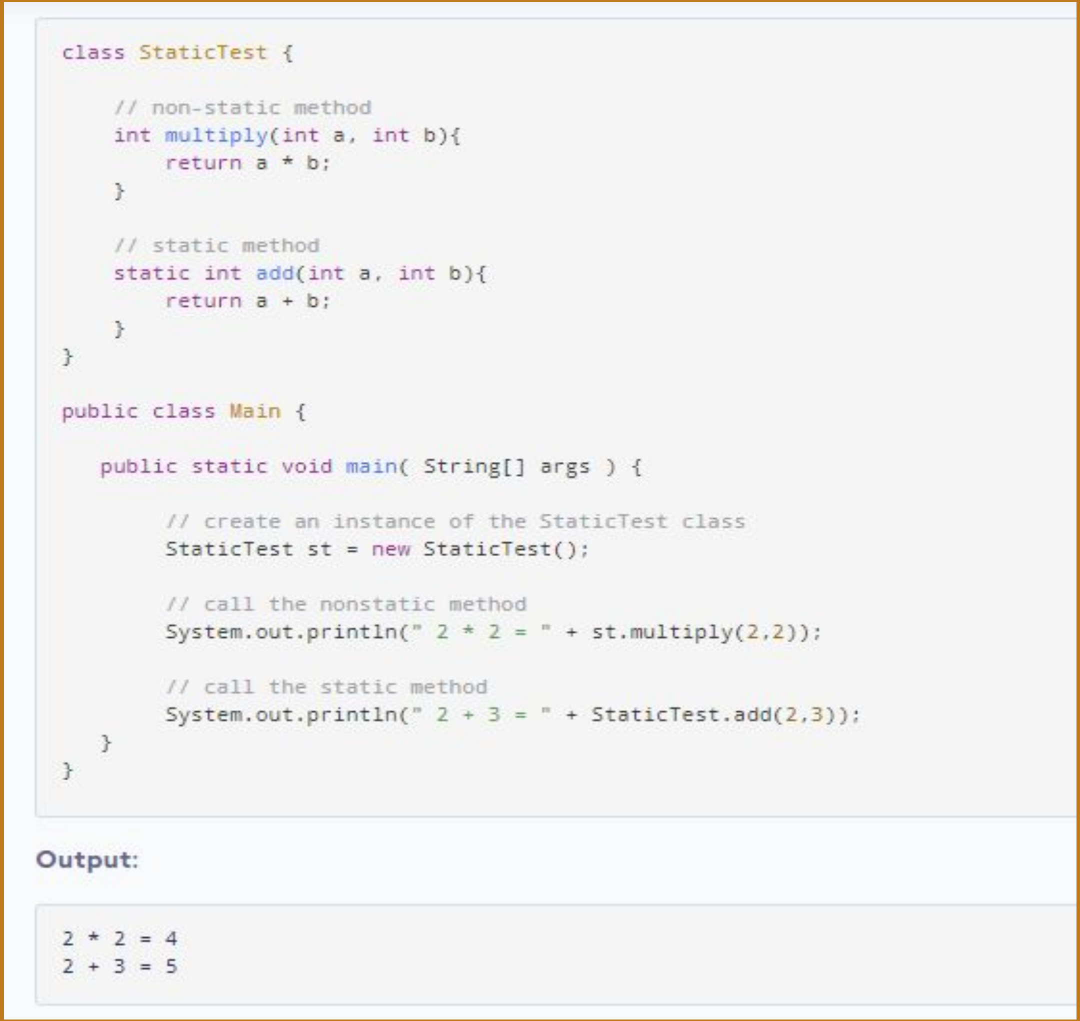
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

STATIC METHOD

- ❖ Static methods are also called class methods. It is because a static method belongs to the class rather than the object of a class.
- ❖ And we can invoke static methods directly using the class name. For example,



```
class Test {
    // static method inside the Test class
    public static void method() {...}
}

class Main {
    // invoking the static method
    Test.method();
}
```



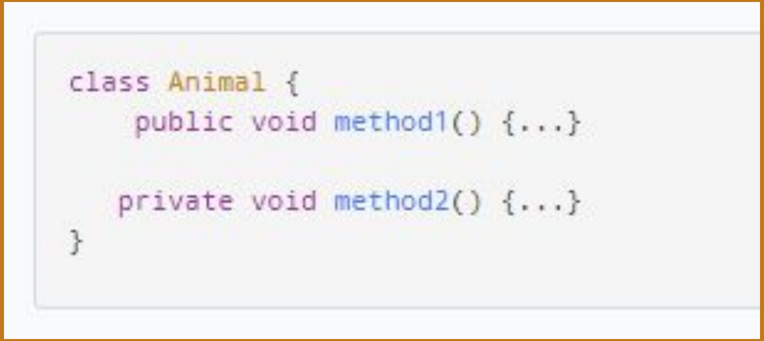
```java
class StaticTest {

    // non-static method
    int multiply(int a, int b){
        return a * b;
    }

    // static method
    static int add(int a, int b){
        return a + b;
    }
}

public class Main {

   public static void main( String[] args ) {

        // create an instance of the StaticTest class
        StaticTest st = new StaticTest();

        // call the nonstatic method
        System.out.println(" 2 * 2 = " + st.multiply(2,2));

        // call the static method
        System.out.println(" 2 + 3 = " + StaticTest.add(2,3));
   }
}
```

Output:

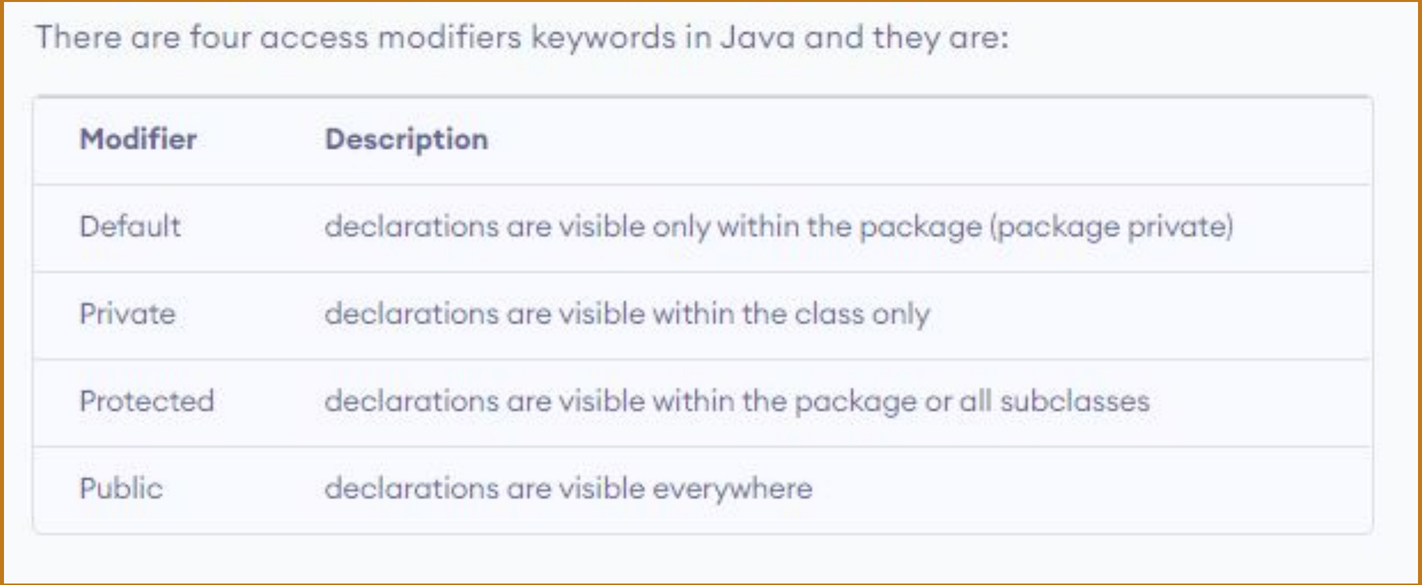
```
2 * 2 = 4
2 + 3 = 5
```

VISIBILITY MODIFIERS

- In Java, access modifiers are used to set the accessibility (visibility) of classes, interfaces, variables, methods, constructors, data members, and the setter methods. For example,
- In the above example, we have declared 2 methods: method1() and method2(). Here,
  - method1 is public - This means it can be accessed by other classes.
  - method2 is private - This means it can not be accessed by other classes.
- Note the keyword public and private. These are access modifiers in Java. They are also known as visibility modifiers.

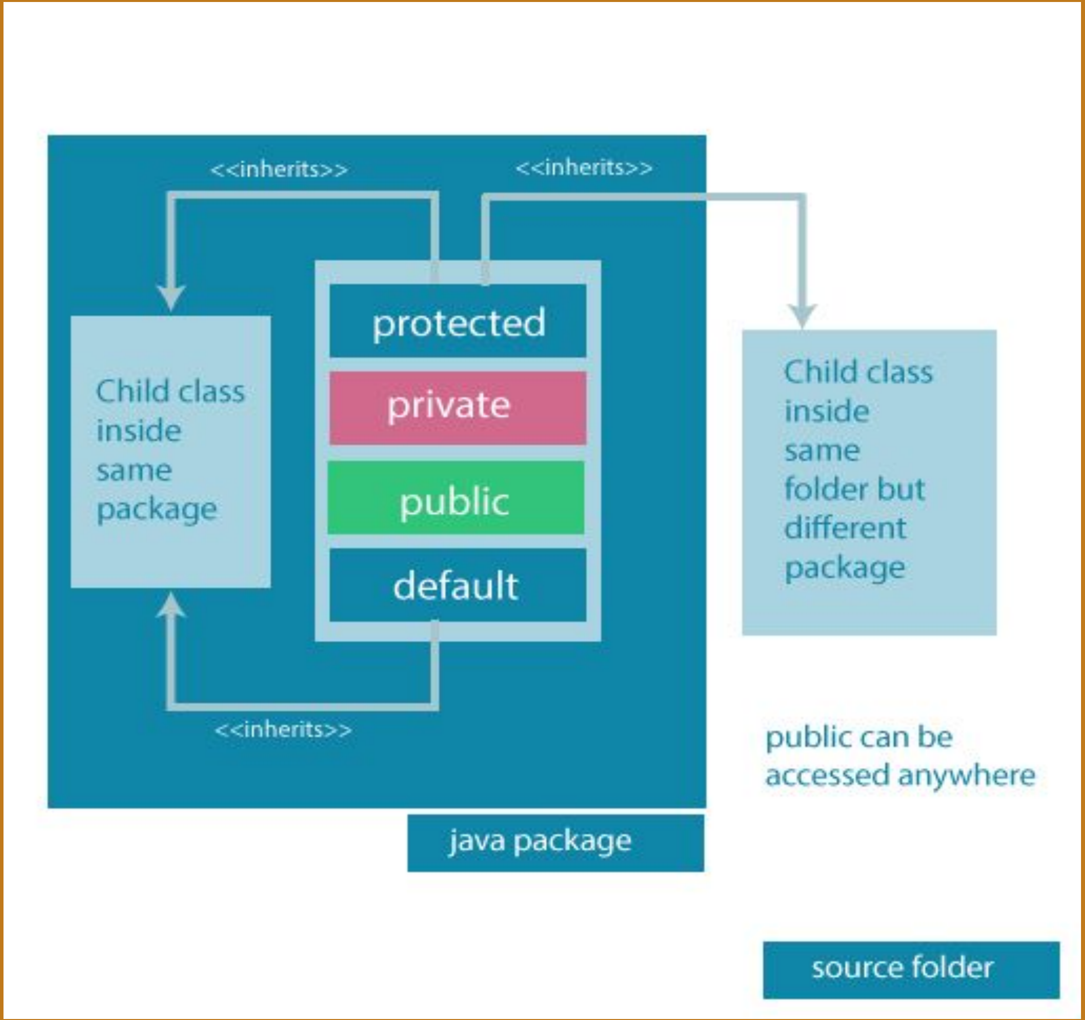


```
class Animal {
    public void method1() {...}

   private void method2() {...}
}
```

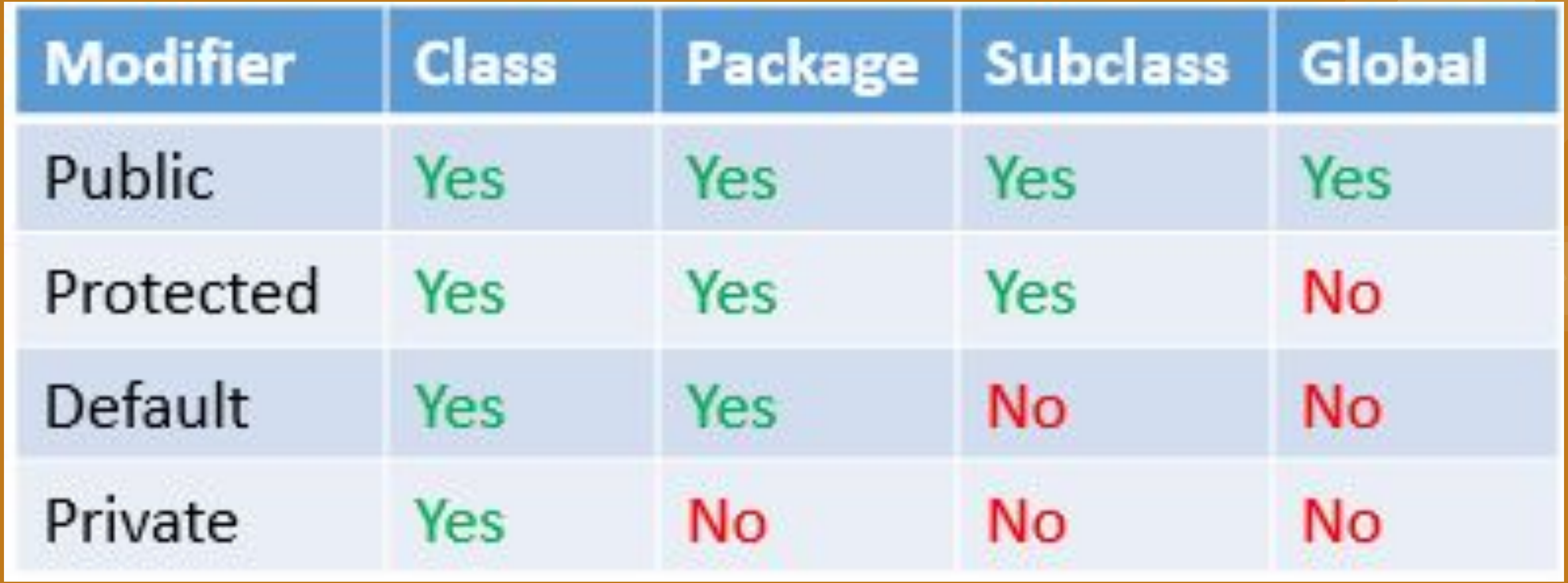


There are four access modifiers keywords in Java and they are:

Modifier	Description
Default	declarations are visible only within the package (package private)
Private	declarations are visible within the class only
Protected	declarations are visible within the package or all subclasses
Public	declarations are visible everywhere



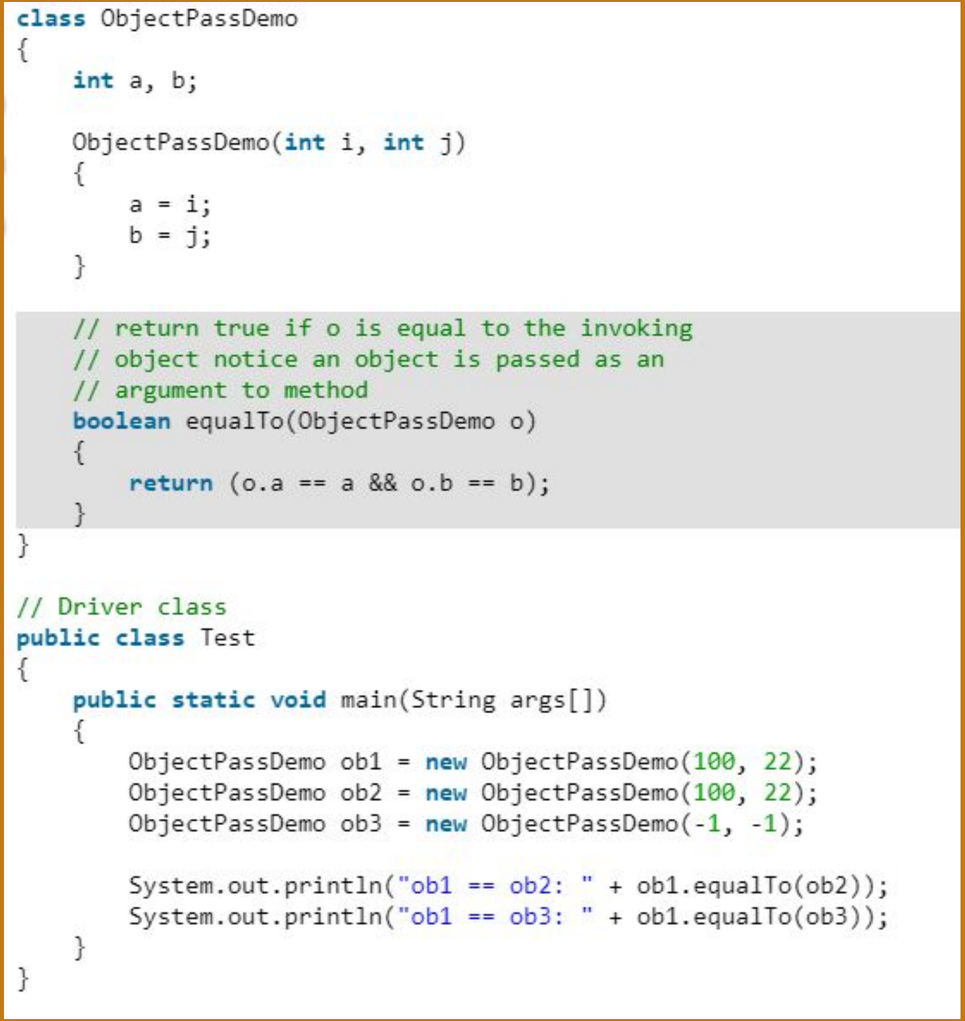
<<inherits>>
<<inherits>>
Child class inside same package
protected
private
public
default
Child class inside same folder but different package
<<inherits>>
public can be accessed anywhere
java package
source folder



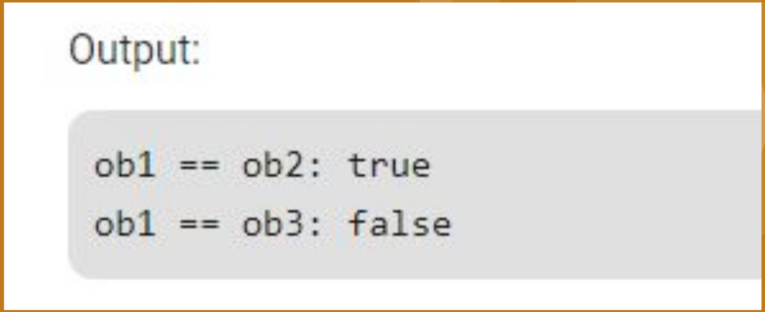
Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

PASSING OBJECTS TO METHODS

- When we pass a primitive type to a method, it is passed by value. But when we pass an object to a method, the situation changes dramatically, because objects are passed by what is effectively call-by-reference. Java does this interesting thing that's sort of a hybrid between pass-by-value and pass-by-reference. Basically, a parameter cannot be changed by the function, but the function can ask the parameter to change itself via calling some method within it.
  - While creating a variable of a class type, we only create a reference to an object. Thus, when we pass this reference to a method, the parameter that receives it will refer to the same object as that referred to by the argument.
  - This effectively means that objects act as if they are passed to methods by use of call-by-reference.
  - Changes to the object inside the method do reflect in the object used as an argument.



```java
class ObjectPassDemo
{
    int a, b;

    ObjectPassDemo(int i, int j)
    {
        a = i;
        b = j;
    }

    // return true if o is equal to the invoking
    // object notice an object is passed as an
    // argument to method
    boolean equalTo(ObjectPassDemo o)
    {
        return (o.a == a && o.b == b);
    }
}

// Driver class
public class Test
{
    public static void main(String args[])
    {
        ObjectPassDemo ob1 = new ObjectPassDemo(100, 22);
        ObjectPassDemo ob2 = new ObjectPassDemo(100, 22);
        ObjectPassDemo ob3 = new ObjectPassDemo(-1, -1);

        System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));
        System.out.println("ob1 == ob3: " + ob1.equalTo(ob3));
    }
}
```

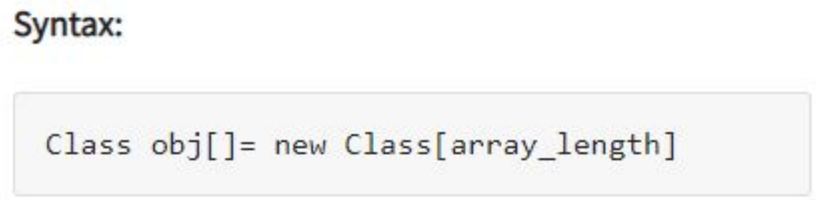


Output:

```
ob1 == ob2: true
ob1 == ob3: false
```

ARRAY OF OBJECTS

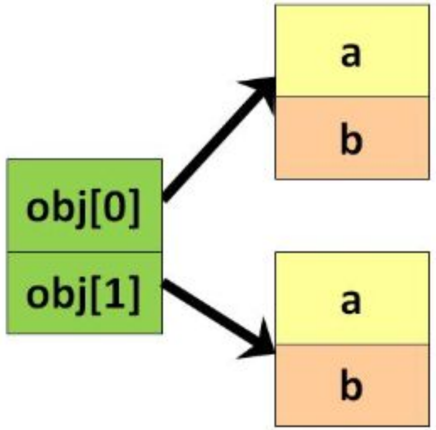
- ❖ **JAVA ARRAY OF OBJECT**, as defined by its name, stores an **array of objects**. Unlike a traditional array that store values like string, integer, Boolean, etc an array of objects stores OBJECTS. The array elements store the location of the reference variables of the object.



Syntax:

```
Class obj[]= new Class[array_length]
```

```java
class ObjectArray{
   public static void main(String args[]){
     Account obj[] = new Account[2] ;
     //obj[0] = new Account();
     //obj[1] = new Account();
    obj[0].setData(1,2);
    obj[1].setData(3,4);
    System.out.println("For Array Element 0");
    obj[0].showData();
    System.out.println("For Array Element 1");
     obj[1].showData();
  }
}
class Account{
  int a;
  int b;
 public void setData(int c,int d){
   a=c;
   b=d;
 }
 public void showData(){
   System.out.println("Value of a ="+a);
   System.out.println("Value of b ="+b);
 }
}
```



Output:

```
For Array Element 0
Value of a =1
Value of b =2
For Array Element 1
Value of a =3
Value of b =4
```

IMMUTABLE OBJECTS AND CLASSES

- **Immutable class** means that once an **object** is created, we cannot change its content. In **Java**, all the wrapper **classes** (like Integer, Boolean, Byte, Short) and String **class** is **immutable**. We can create our own **immutable class** as well. ... The **class** must be declared as final (So that child **classes** can't be created)

```java
// An immutable class
public final class Student
{
    final String name;
    final int regNo;

    public Student(String name, int regNo)
    {
        this.name = name;
        this.regNo = regNo;
    }
    public String getName()
    {
        return name;
    }
    public int getRegNo()
    {
        return regNo;
    }
}

// Driver class
class Test
{
    public static void main(String args[])
    {
        Student s = new Student("ABC", 101);
        System.out.println(s.getName());
        System.out.println(s.getRegNo());

        // Uncommenting below line causes error
        // s.regNo = 102;
    }
}
```

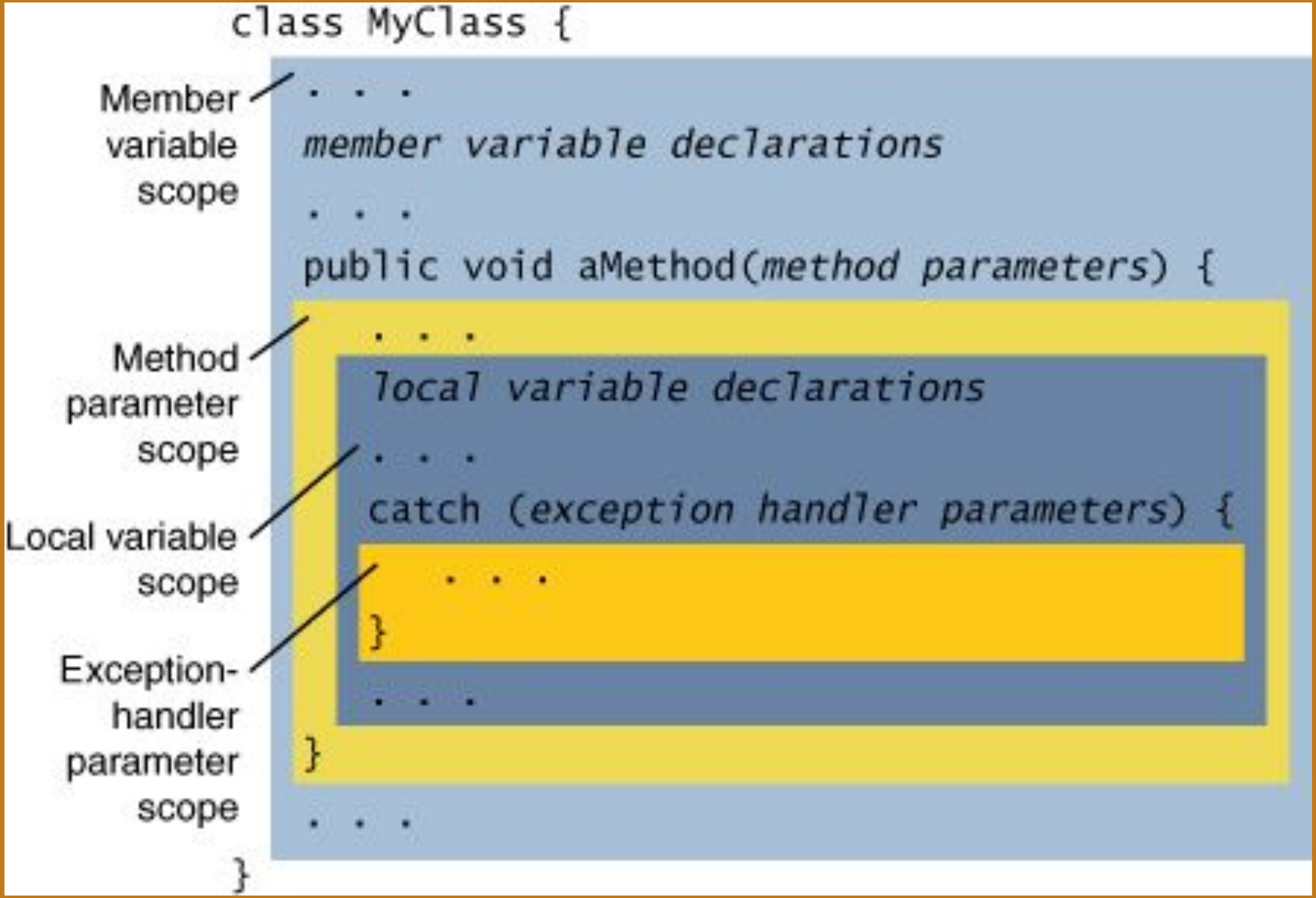
Output:

```
ABC
101
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

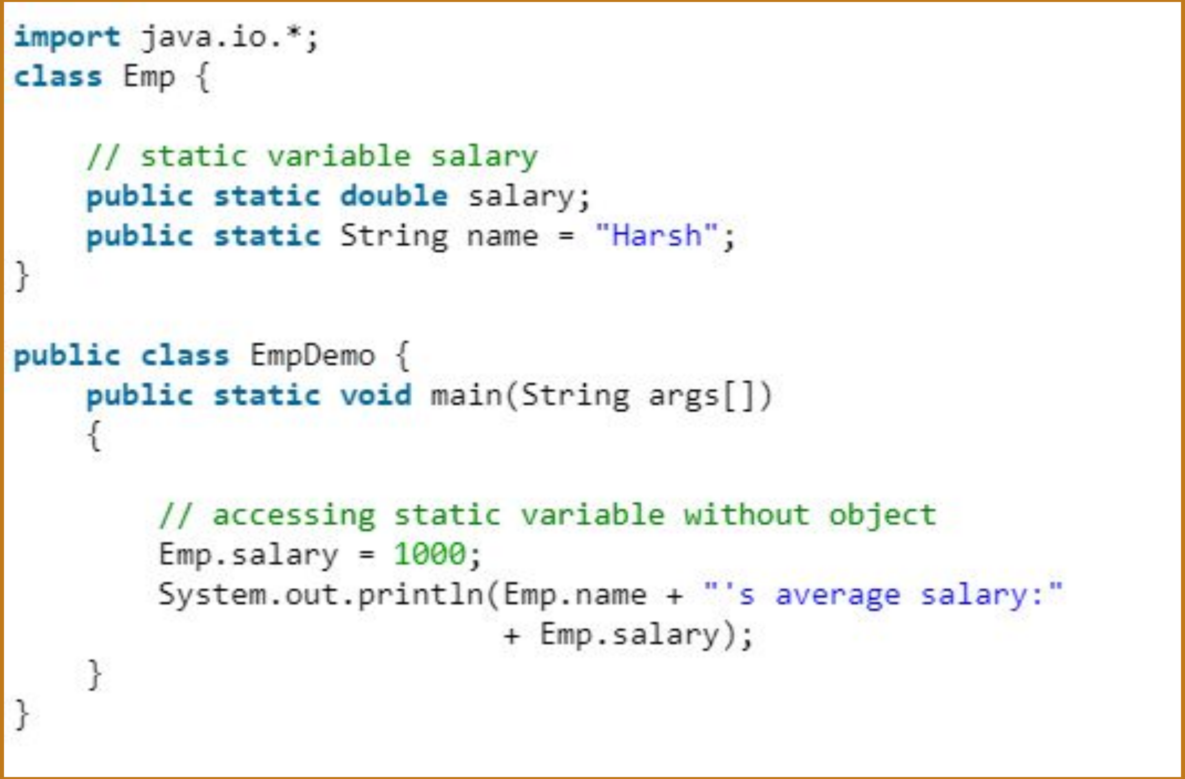
SCOPE OF VARIABLE

- By default, a variable has default access. Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.
- A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.
- Java provides a number of access modifiers to set access levels for classes, variables, methods, and constructors. The four access levels are -
  - Visible to the package, the default. No modifiers are needed.
  - Visible to the class only (private).
  - Visible to the world (public).
  - Visible to the package and all subclasses (protected).

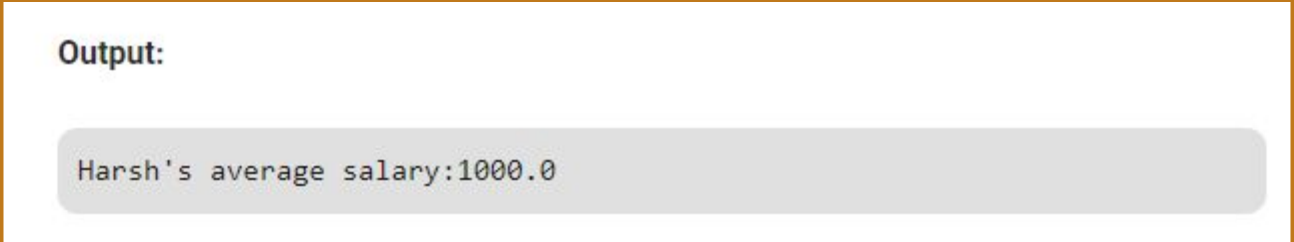


```
class MyClass {
    . . .
    member variable declarations
    . . .
    public void aMethod(method parameters) {
        . . .
        local variable declarations
        . . .
        catch (exception handler parameters) {
            . . .
        }
        . . .
    }
    . . .
}
```

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY



```java
import java.io.*;
class Emp {

    // static variable salary
    public static double salary;
    public static String name = "Harsh";
}

public class EmpDemo {
    public static void main(String args[])
    {

        // accessing static variable without object
        Emp.salary = 1000;
        System.out.println(Emp.name + "'s average salary:"
                                   + Emp.salary);
    }
}
```



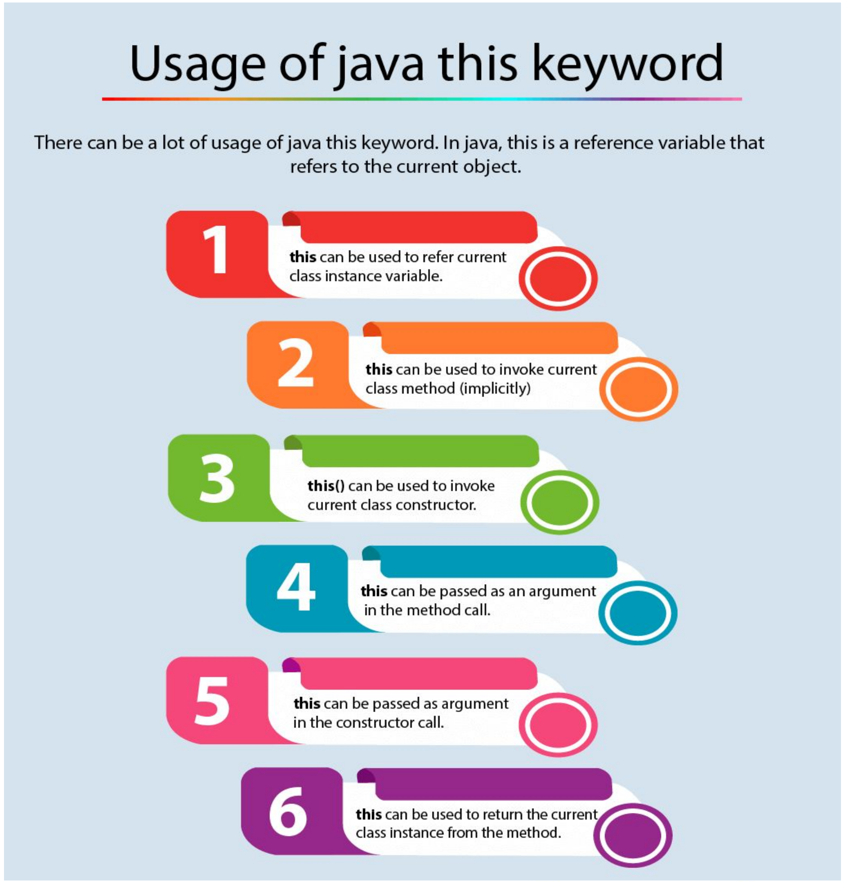
Output:

```
Harsh's average salary:1000.0
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

THIS REFERENCE

In Java, this keyword is used to refer to the current object inside a method or a constructor. For example,

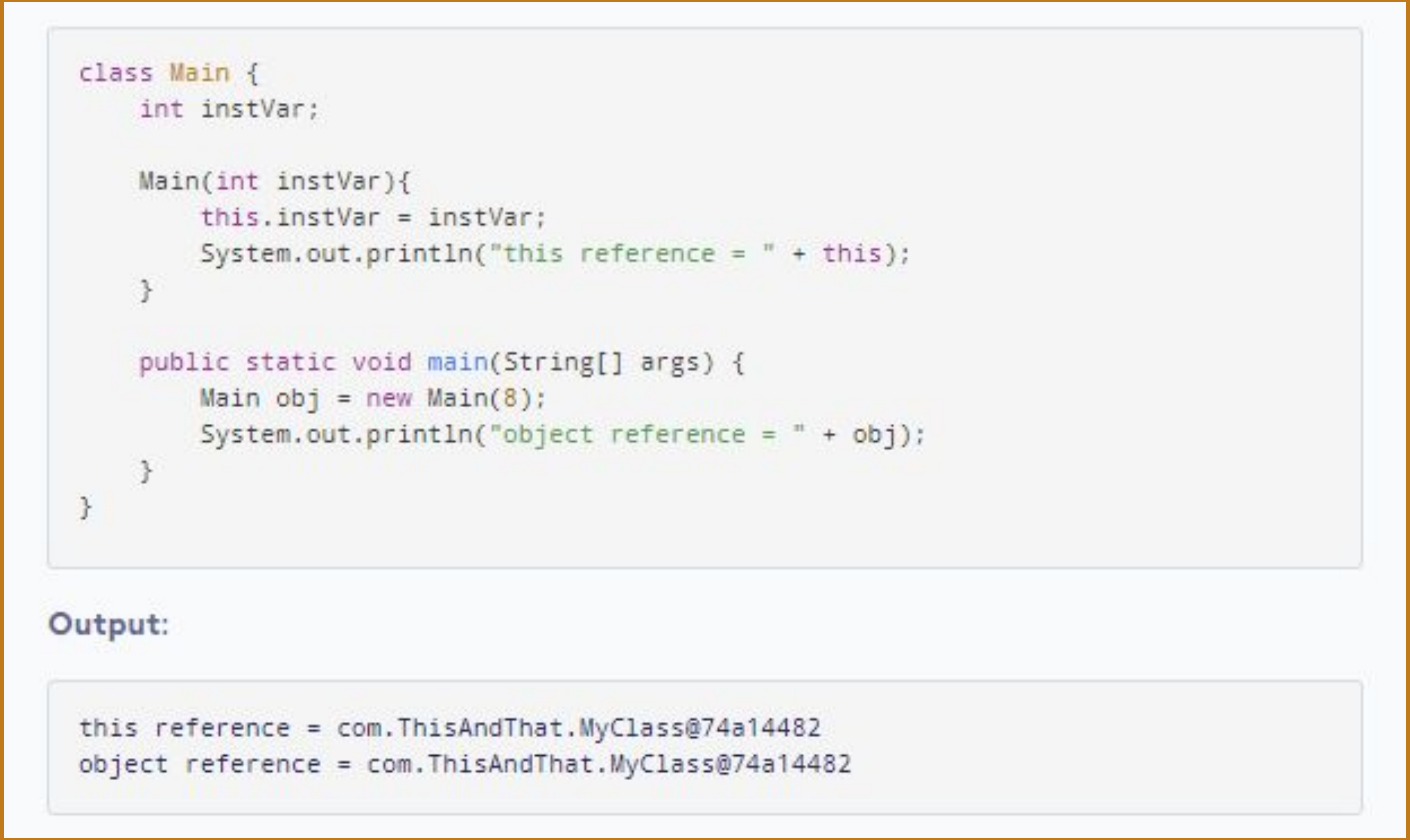


Usage of java this keyword

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

1. **this** can be used to refer current class instance variable.
2. **this** can be used to invoke current class method (implicitly)
3. **this()** can be used to invoke current class constructor.
4. **this** can be passed as an argument in the method call.
5. **this** can be passed as argument in the constructor call.
6. **this** can be used to return the current class instance from the method.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY



```java
class Main {
    int instVar;

    Main(int instVar){
        this.instVar = instVar;
        System.out.println("this reference = " + this);
    }

    public static void main(String[] args) {
        Main obj = new Main(8);
        System.out.println("object reference = " + obj);
    }
}
```

Output:

```
this reference = com.ThisAndThat.MyClass@74a14482
object reference = com.ThisAndThat.MyClass@74a14482
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY